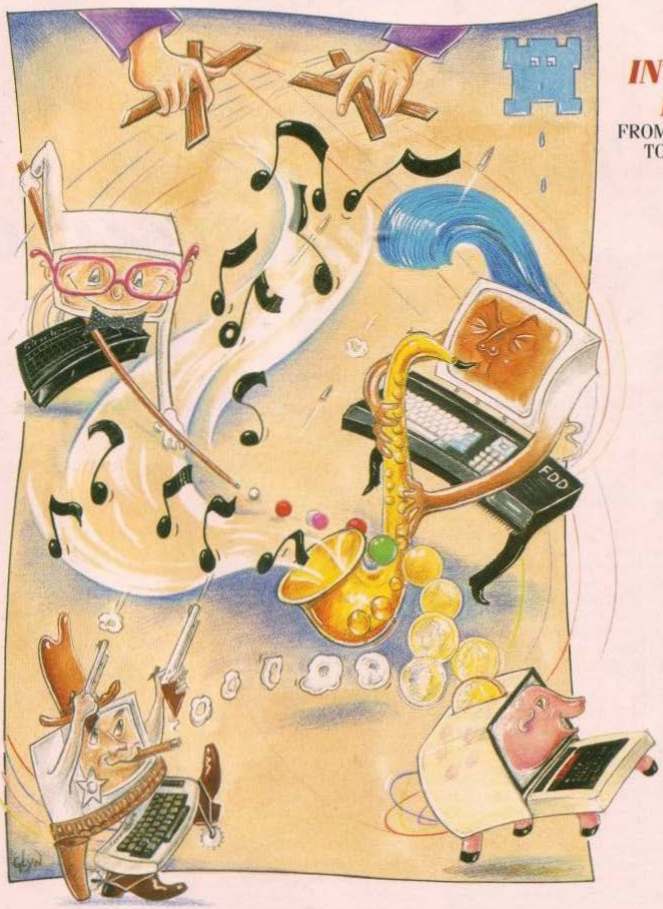
Your COMPUTER COURSE

MARCH 1986

A COMPREHENSIVE GUIDE TO PROGRAMMING



INTRODUCING LANGUAGES
FROM FORTH PROTOCOLS TO BASIC STATEMENTS

A-Z OF COMPUTER JARGON
PARLEZ-VOUS COMPUTER SPEAK?

MACHINE CODE CRACKED
THE MYSTERIES REVEALED

MUSIC & GRAPHICS
CREATING SOUND AND VISION!

Welcome to the *Your Computer* course, a six-part series which will put you in the front-line of programming and using your micro. We will show you how to program in Basic, which language to choose if you become disenchanted with Basic, how to program in machine code, how best to make use of sound, graphics and music on your micro, and finally how to deal with the vast library of jargon which festoons the microcomputer business.

The series will be offered in collectable monthly parts, so that you can build your course, following each section as its printed. We will also offer a directory for the course in the issue following the publication of the last part, so that you can find your way easily round the finished course.

We hope you enjoy the course and find that it adds considerably to your fund of knowledge about programming. We have tried to keep it simple and easy to understand, without making it so basic that it repeats what you have in the programming manual you receive with your machine.

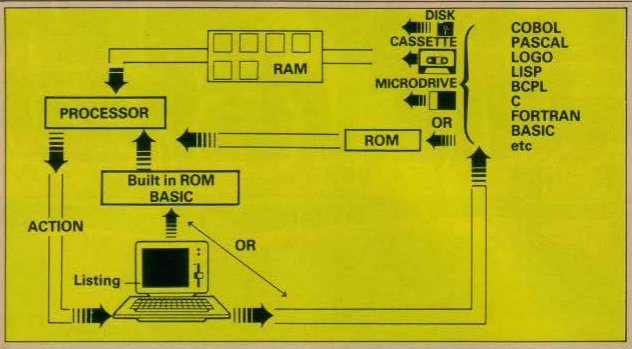
It is assumed that you will have had at least a passing familiarity with Basic, as laid down in your machine documentation, and that you have some interest in programming. Beyond that, we ask only that you read and enjoy.

So sit down, relax and get ready for a series of entertaining lectures. The first class begins opposite.

INDEX

Languages 2
Glossary 4
Machine code 6
Sight & Sound 8
Next Month 8

Contributors:
Geoff Wheelwright and David Janda.



Tools of the programmer's trade

Basic was designed to be an all purpose programming language for students to learn computing. It is still by far the most popular program language used on micros, but by no means the best.

Computer languages are really tools to help you do the job of operating your computer – and Basic is just one of a wide range of popular programming tools available to you. In the next six issues, we will be looking at the screwdrivers, hammers and jigsaws of the programming world – a world in which Basic could be compared to a pocketknife; useful for many small jobs, but not something you could stand working with all the time.

The major criteria in examining alternatives to Basic has to be the types of programs you plan to write. Some programs are fast at sorting databases, while others excel at maths calculations. Certain languages provide you with more precise control over any instruments you may have hooked to your machine and yet others are designed to be easily-modified.

Prolog, for example, is used for data-base management purposes because it has very flexible data types. Comal, on the other hand is ideal for those who wish to learn a more structured language than Basic, yet retain some of the finer points of the latter. If process control or very fast processing is required, then Forth is the ideal language. If scientific and mathematical problems are to be tackled, Pascal is the obvious choice.

Even if you are not involved in a specific area of computing, it is still worth learning a new programming language if only for the fun of it. To give you a taste of the diverse character of computer languages, we begin our series with some brief profiles of four interesting languages and dialects.

Prolog – Spectrum, C64, BBC model B.

Prolog – PROgramming in LOGic – is becomming one of the most popular programming languages. Used extensively in the field of artificial intelligence, it is one of the fifth-generation computer languages.

Conventional languages are not very flexible when it is a matter of describing the problem to be solved. It would be far better if it were possible to describe the problem in general terms by describing the problem in an English-type manner. Providing the description contains all the data necessary to solve the problem, it should, in theory, be possible for a computer to find the answer.

Computer languages are tools to help you with the operation of your computer. Without them the machine will not function.

A declarative type of language such as Prolog can, to an extent, achieve this. A Prolog program consists of a set of rules and facts which make up a program description. Both facts and rules are referred to as sentences which must be written in a precise form, known, not surprisingly, as a sentence form. Any questions put to the program are called queries.

Prolog, the core language, is a little difficult to understand and learn, so a user-friendly front-end called Simple has been developed to make Prolog easier for beginners to use.

Here is a mini-database of facts and rules relating to a group of people:

likes (Diane Colin)

```
likes(Janet Ian)
smokes(Ian)
smokes(Jean)
enjoys(Diane rock)
enjoys(Jean reggae)
gives(Sam Diane gin-and-
   tonic)
gives(Sam Colin cola)
```

The structure for Prolog facts is as follows. First, there is a predicate such as 'likes' which is followed by a bracketed list of terms called arguments. Predicates correspond roughly to verbs in English and arguments to nouns. Here is a more complex sentence:

```
partners(Diane Sam) if
   gives(Sam Diane gin-and-
   tonic) &
   not smokes(Sam)
```

With the information stored in the micro, an IS query can be used to ask simple questions:

```
is(likes)Janet Ian))
```

YES will be displayed. IS simply finds out whether a relationship is true or false. Another example is:

```
is(smokes(Janet))
```

NO will be displayed.

A more complex question can be asked:

```
is(partners(Diane Sam))
```

This time Prolog has to do a little more work. Prolog had to find whether Sam gave Diane a gin and tonic AND didn't smoke! YES and NO are not very informative answers.

A WHICH query uses variables in place of data objects:

```
which(x : smokes(x))

Ian
Jean
No (more) answers
```

. . would be displayed. Prolog is asking which term could replace x in the goal 'smokes(x)' to make the goal succeed.

Comal – C64, BBC model B

In the early days of computing, Basic was not accepted by all as the answer to everything. People realised that it had many faults and that it could be improved. The structure and syntax of Basic was often criticised and because of that Comal was invented.

Comal and Basic are very similar in many respects and that helps the Basic user to progress to Comal very easily. There are, however, significant differences between the two, and once Comal is mastered it is easier to progress to more modular languages such as Pascal.

Comal allows a programmer to write code in a modular fashion. The dreaded GOTO is not allowed; instead, labels can be used to jump to a specified section of the program. Procedures are fully-supported with the use of EXEC. Using EXEC with parameters allows a call to a PROCedure which can have local variables assigned to it.

A case statement is incorporated which is similar to ON GOSUB, although more flexible. The FUNC command allows multiple line functions to be defined, unlike the Basic equivalent where single-line functions are permitted.

LIST will produce automatically an indented listing which is easy to read. Comal also 'fills' in pieces which the beginner may omit, consider the following:

```
10  FOR A=1 TO 1000
20  B=A
30  NEXT A
```

will list in Comal as:

```
0010  FOR A:=1 TO 1000 DO
0020  B:=A
0030  ENDFOR A
```

Forth – all major home computers

Of all the programming languages available, Forth must be one of the most popular. It has been chosen by many manufacturers as the second language for their machines. Part of the reason is the vast amount of Forth public domain software and so anybody can implement it on a machine, without the worry of being sued.

Forth was invented by Charles H Moore in the late 1960s and early 1970s and was designed primarily as a control language for telescopes. Two main versions of Forth are in use, Fig-Forth which was developed by the Forth Interest Group (FIG) and Forth-79. An updated version of Forth-79 called Forth-81 is also available.

Many describe Forth as the high-level-low-level language. That is because Forth has high-level control constructs yet operates very close to machine level, to the extent that you have to manipulate a stack when using the language. The fact that Forth commands (words) do not relate directly to what happens is a little confusing when looking at a Forth listing. As an example, the Forth word '.' will print a number which is on the top of the stack but you would not gather that from looking at the word.

Forth, however, is very powerful. So far as speed is concerned, Forth is almost as fast as machine code and that is another reason for its popularity. The biggest asset of Forth is that the programmer is not tied to any set commands.

Forth words are held in what is called the dictionary, which can be added to. This is done by defining a new word:

```
: GREETING ." Hello" ;
```

Here, a new word called GREETING has been added to the dictionary and whenever you need to print 'Hello' you simply enter GREETING.

That can be used within other Forth words, so if you wanted to define a word to print 'Hello' one hundred times, the following would do it:

```
: WELCOME 100 0 DO
  GREETING LOOP ;
```

It should be noted that the new definitions are normally written using an editor, saved to tape or disc, and loaded with the Forth at the start of each session.

Languages are the hammers, screwdrivers and jigsaws of the programming world.
As ever, it is important to choose the right tool (language) for the task in hand.
Basic, for example, is ideal for many tasks but unsuited to games programs.

Pascal – All home computers

Pascal, like Forth, was designed by one man and not a committee. Nicklaus Wirth, the inventor of Pascal, designed the language to be used as an aid to teach programming. Pascal has been taken up in business and scientific areas and it is the language taught in many colleges and universities.

The benefits are that it is supposed to promote good programming practice. Programmers are almost compelled to write code which is organised into logical segments. That is the theory and, in practice, people who learn the language write programs which are easier to understand and debug.

Unlike Basic, Pascal permits users to define their own data types. That means that, as well as integers, characters and real numbers, more can be defined. A data type consisting of the pieces on a chess board is one somewhat obscure example.

The structure of a Pascal program is rather strange to Basic programmers. Pascal requires groups of commands to be embedded in blocks within the Pascal statements BEGIN and END. Procedures and functions are fully-supported and can reference each other freely.

Pascal requires all identifiers (variables) to be declared at the beginning of the program, as well as arrays, constants, data types and records. To Basic programmers that is horrible, because it means they have to know what identifiers are used before the program is written.

It is difficult at first to become accustomed to the language but once the disciplines have been accepted, better programs can be written.

```
PROGRAM Demo (INPUT,
   OUTPUT);
VAR
   I : INTEGER;
BEGIN
   FOR I:=1 TO 100 DO
      WRITELN(I,I*I);
END.
```

Most home computers will have versions of the foregoing languages available for them. Make a thorough search when buying a language package, as there will probably be more than one version of the language available for your micro. Be prepared to buy some tutorial books as well, as the manuals are not always well-written.

A

Computer jargon has become the bane of many us boffins, ensuring that the level of understandin concerned about the linguistic integrity of the la to bring together terms, both frivilous and funda cover letters A to D and acquaint you with Lettic

■ **Algorithm:** A mathematical and/or logical procedure used to solve problems. See **Bug.**

■ **Alphanumeric:** 'Has alphanumeric keyboard' is a flash way of saying it has all the letters of the alphabet and numbers 0-9 on it. Beware of non-alphanumeric keyboards.

■ **Analogue to digital conversion:** Computers think in numbers (digits) while broadly the world continues with analogue scales, so you cannot say things like "He smelled fively." To have a computer do so – and they do – you need to convert the analogue scale to digital.

■ **Analyst:** The role of analyst is of critical importance in the computer industry and is defined by many businesses as someone who examines a manual system which doesn't work, replaces it with a computer system which doesn't work but costs you £X,000, then charges you £100 an hour for his time. Theoretically, an analyst defines problems, then develops systems for solving them.

■ **APL:** A Programming Language. Like all programming languages of which you heard but never used, APL is powerful and easy to learn. Like all those languages you will probably never have to use it.

■ **Applications software:** Computers are supposed to make it easier for you to perform specific tasks, or applications, so the programs a computer runs to do those tasks fall under the heading of applications software. The best applications software does not work at all. Beware of that which works it will become apparent that it does not only after you have lost several thousand pounds.

■ **Architecture:** Nothing to do with office blocks. The way a microprocessor is designed, the architecture governs organisation, location and amount of temporary storage, processing capacity and so on.

■ **Arithmetic:** The process by which some micros multiply 2 by 2, divide the sum by 2 and get 1.99999. If you understand why they do it you probably wish you did not.

■ **Array:** A computer stores information in an array. Imagine you are postboy for a large company. Your array is a big bank of pigeonholes, into which you put the mail, or information. A table showing the multiplied values of number 1x1 to 12x12 is also an array.

■ **Artificial intelligence:** It is arguable whether or not computers think but they certainly do not reason, so while they can calculate till the cows come home, they are incapable of that wild, off-the-wall stroke of genius you once had/might have/wish you could have. The industry, however, is working on this one and true artificial intelligence will mean that computers will be able to hold ordinary jobs and/or even become President of the United States.

■ **ASCII:** The American Standard Code for Information Interchange, naturally enough, emanates from the ASCII Corp in Japan. ASCII assigns a number to each of the characters a normal keyboard can produce so that, for example, when a computer says to a printer 'print 32' the printer knows that 32 is a space. Otherwise the computer would have to spend time describing what a space is.

■ **Assembler:** A way to convert a Basic program which works slowly into a fast machine code program which does not work. See **Machine code.**

B

■ **Background task:** While you are typing at a word processor you may want your computer to be doing something else. A program which does that while you are working in the foreground is performing a background task. See **Multi-tasking.**

■ **Back space:** It has nothing to do with your back garden. Backspace is the key which might delete the character immediately to the left of where you are on the screen, or it might move you on top of that character. See **Standardisation.**

■ **Back-up:** Originally to take a security copy of a program or file. Now used interchangeably with pirate and steal.

■ **Bar code:** One of those stripey things you see on a tin of beans. It is there because computers have difficulty reading difficult things like "Now only 23 pence a tin." Supermarket checkouts cannot usually read bar codes, either, but occasionally you see one which can on *Tomorrow's World*. It will happen one day.

■ **Basic:** Beginners' All-purpose Symbolic Instruction Code, the programming language run by the vast majority of microcomputers. Some say it is like English but either they are wrong or they say things like: FOR N=1 to 255 : PRINT N;",";CHR$ (N) : NEXT It is much more like English than other programming languages but you should not be carried away.

■ **Baud:** Does not mean bits per second but for all the difference it makes it might as well do so. Baud is a way to measure the speed at which information can be sent, hence bits per second. See **Bit.**

■ **Benchmark:** As used by the Ordnance Survey, a benchmark is a standard height mark against which other heights can be judged. The term is used by computer manufacturers to provide sometimes misleading information, based on tests.

■ **Binary:** Fred Flintstone, cave-dwelling father of modern arithmetic, had 10 fingers, hence decimal. Computers have only two fingers, on and off, and so count zero, one, ten, eleven, one hundred.

■ **Bit:** Short for a binary digit. A bit is the smallest element with which a computer can deal and can have one of two values – 0 or 1. See **Binary.**

■ **Black box:** Any piece of hardware which does something you don't understand – but does it properly – is a black box.

■ **Boolean:** A form of logic used in programming. Not widely understood.

■ **Boot:** Often used to mean 'switch on' but of computers should more properly mean 'switch on and wake up'.

■ **Bootstrap:** A program which tells the computer what to do when it wakes up, i.e., print 'Hello, where am I? How do I feel this morning . . . uh . . . READY>'

■ **Bottom-up programming:** A method of programming which concentrates on the detail and crosses its fingers that the bits will all work when they are fitted together. Much derided.

■ **Bubblesort:** A method of sorting where pairs in a series are swapped until the whole series is in order.

■ **Buffer:** Member of a Whitehall gentleman's club. Alternatively, a way to store data for a short time where the originating machine works faster than the receiving machine.

■ **Bug:** When your computer tells you 2+2=Tuesday you have a bug in your program. See **Feature.**

■ **Bundle:** A method of selling unsaleable software by including it with a computer which otherwise would be poor value.

■ **Bus:** A configuration of circuitry ostensibly meant to pass information between one part of a system and another. An expansion bus is a plug which if you could get the proper cable – which you can't – would allow you to connect your computer to

ers. Often new terms creep quietly into the daily language of g among most people remains low. At Your Computer we are nguage and have thus assigned language expert John Lettice amental, to make up the following glossary. In this issue, we e's idiosyncratic view of the computer world.

add-ons, if they were available, which they're not.

■ **Byte:** Eight bits. See **Bit.**

■ **C:** Possibly the computer industry's favourite programming language. See **Techie.**

■ **Cable:** When you've just bought a computer you'll find a cable is the one piece of equipment you need which isn't included. Alternatively, a cable is the piece of equipment included instead of the proper cable.

■ **CAD:** Computer Aided Design. Industry pros wax lyrical on the potential of this area. If involved in a conversation, use the terms 'Kray', 'Max Headroom', 'gigabyte of memory'.

■ **Card:** A piece of paper listing the permanently-engaged telephone number of the guy who sold you the computer which doesn't work. Alternatively, a card is a crucial piece of your computer which was not included when you bought it, allowing the manufacturer to sell you something else. See **Expansion slot.**

■ **Cartridge:** A small slot-in piece of hardware which allows you to turn your micro into a games console – worth half the price of your micro.

■ **Cassette:** The cheapest way of storing programs. So you have only yourself to blame when it doesn't work.

■ **Character:** Jack Tramiel. Alternatively, a letter, number or indescribable blob which appears on your micro screen.

■ **Character recognition:** Saying "Jack Tramiel is a force to be reckoned with." Alternatively, a way to make a computer read a paper document and understand it as q pwqer documitt. Optical scanning is not all it might be . . .

■ **Character set:** Jack Tramiel's family. Also all the letters and numbers your micro can handle. A character set is always missing one character (e.g., £) which you need desperately.

■ **Chip:** This is what it's all about. The current generation of computers runs on chips made of sand. Lawrence of Arabia was sitting on a gold mine.

■ **CMOS:** Complementary Metal Oxide Semiconductor, a flash chip which draws much less power than standard chips and therefore is used in battery-powered machines.

■ **Cobol:** Computer language used in business programming.

■ **Command:** Used to instruct computers. Usually elicits the response 'Syntax error'.

■ **Communications:** First drunk: "Is this Wembley?" Second drunk: "No, it's Thursday." First drunk: "So am I, let's have another drink." Imagine two computers having that conversation and you understand communications.

■ **Computer:** A device used to automate manual functions so that they take more time and cost more to perform. Whatever its size, a computer will occupy more space on your desk then you have available.

■ **Conditional branch/jump:** The point at which you lose track of what the program you have written is doing.

■ **Control characters:** Unidentified garbage on the screen.

■ **Conversion:** A method used by games software producers to make a Commodore 64 look like a Spectrum or, alternatively, to make a 64 program a mess when running on the Spectrum.

■ **Co-processor:** When a manufacturer has made a great deal of money by selling you a machine with an under-powered CPU, it can make even more money by selling you a more powerful CPU, or co-processor. See **CPU.**

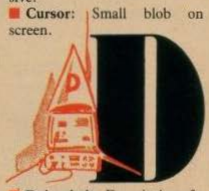
■ **CP/M:** Sometimes claimed to stand for Control Program/ Microcomputer but also sometimes claimed to stand for something else. Until recently CP/M was something which was the industry-standard operating system for micros and that was all you needed to know. Since Amstrad machines generated a CP/M revival it has also become necessary for you to know what CP/M is. See **Operating system.** See **Industry standard.**

■ **CPS:** Characters per second. A poorly-calibrated scale used by printer manufacturers to produce a figure 25-30 percent greater than the maximum speed their printers can achieve.

■ **CPU:** Communist Party of Yugoslavia to those in the know but in the micro world it stands for central processing unit. The CPU is the main chip inside the computer.

■ **CRT:** Cathode ray tube, the central component of a television set or monitor. Television sets have more components and are therefore cheaper. Monitors have fewer, so are more expensive.

■ **Cursor:** Small blob on screen.



■ **Daisychain:** Description of a group of peripherals linked in a series. To make it work you must take it apart, check all the connections and put it back together again. It still won't work.

■ **Daisywheel:** A printer head which has letters arranged round it like daisy petals. Slow, noisy and generally missing one essential character. See **Character set.**

■ **Data:** Forget chips. This is really what it is all about. Data is the information manipulated by a computer program, then stored in a form which proves to be irretrievable.

■ **Debug:** Ten percent of computer programming involves writing, the rest debugging. See **Bug.**

■ **Default:** Most computer errors are default of the operator. Default values are pre-programmed parameters – governing printer line spacing, for example – set to operate if you refuse to answer the computer's questions.

■ **Delete:** See **Back space.**

■ **Demodulator:** The part of a television set which makes it cheaper than a monitor. See **CRT.**

■ **Device:** See **Black box.**

■ **Diagnostic:** Program used by a computer to identify faults. If your micro is not working, don't consult your dealer, run the diagnostics disc. That will confirm that it isn't working and advise you to contact your dealer.

■ **Directory:** A list of files on a disc, generally missing the one you want.

■ **Disc:** Piece of plastic which allows you to store almost as many files as you need to store. See **Winchester, Hard disc.**

■ **Disc drive:** Vent on front of computer, used for trapping dust, cigarette ash and so on.

■ **DOS:** Disc Operating System. Used by computer to control the flow of dust and cigarette ash to the disc drive.

■ **Dot matrix:** Dot matrix printers used to be of poorer quality than daisywheels but miniaturisation and Japanese join-the-dots technology has changed all that. Today's dot matrix printers produce outputs almost indistinguishable from those of a standard typewriter.

■ **Double density:** A kind of disc which will store twice as much information as a standard disc.

■ **Double precision:** System which uses more memory than you can afford to allow a computer to multiply 2 by 2, divide by 2 and get 2. See **Arithmetic.**

■ **Dumb terminal.** Expensive monitor with keyboard attached, communicating with a real computer which is somewhere else. See **CRT.**

Talk the langua of your micro

Home computers have one major disadvantage when you compare them to mainframe computers – they are slower. No doubt you have played video games on your micro and realised that you could not write a Basic program which could match the speed or even achieve the same effects as the game. It is not your lack of programming knowledge which is at fault but the language you are using – Basic.

Machine code is the native language of the microprocessor in your micro. Whenever you turn on your micro the processor is busy setting-up the computer for your use; checking the memory, performing various tests, clearing the screen, looking to see what other devices – printers, disc drives and the like – are attached to the machine. That normally takes about one-tenth of a second.

Even when you are not running a program the central processing unit is executing thousands of machine code instructions per second. When you use Basic, which is written in machine code, you are using a translator of a kind.

The program you have written is scanned by the Basic interpreter and translated into many individual machine code programs.

To find what machine code looks like, look inside the instruction manual for your micro and find the number of the memory location which is at the start of the Basic ROM in your machine. Enter this small Basic program, run it, and when prompted enter the number:

```
10 INPUT A
20 FOR C=A TO A+256
30 PRINT PEEK(C)
40 NEXT C
```

If you have the number of the first memory location (start address) of the Basic ROM in your micro, the screen will display numbers ranging from 0 to 255 – that is machine code.

So far as the CPU is concerned, machine code and data is represented in binary, or base 2:

Decimal	Binary	Hex
1	00000001	01
2	00000010	02
3	00000011	03
4	00000100	03
5	00000101	05
6	00000110	06
7	00000111	07
8	00001000	08
9	00001001	09
10	00001010	0A
11	00001011	0B
12	00001100	0C
13	00001101	0D
14	00001110	0E
15	00001111	0F

When POKing or PEEKing to and from memory from Basic, usually you use decimal.

When you use an assembler – a program to facilitate writing machine code programs – hex normally is used. An understanding of the binary numbering system is essential, as all mathematics in machine code are based on this system.

Programming in Basic involves tasks such as assigning values with the LET statement,

Machine code is the native language of the microprocessor in your computer. It can only understand m/code.

performing calculations on variables, and issuing commands such as PRINT. Many of those things are the same for machine code programs, except that they are done in a different way, depending on the processor in your computer.

For example, the Z-80 is a general-purpose CPU used by many micros, including the Amstrad range and the ZX Spectrum/+. The CPU has 14 registers which can be compared to the Basic equivalent of variables.

Unlike Basic, however, where variables can be used freely by the programmer, CPU registers have special functions assigned to them.

Z-80 CPU registers

Accumulator	A F Flag
General purpose registers	B C D E H L
Interrupt	I R Refresh
Index registers	IX IY
Stack pointer	SP
Program counter	PC

There are six general-purpose registers for use by the programmer. One of the most important registers is the accumulator 'A', which is used for most calculations. Each register (or variable) can store a value between 0 and 255. Basic variables can be assigned in different ways and so can CPU registers. One way is to assign the accumulator a numeric value. If you want to load the accumulator with the number seven, the instruction and data will look like this in binary:

```
00111110
00000111
```

That is rather confusing. In Hex it would be:

```
3E
07
```

That is a little clearer but if we use an assembler, the instruction would be:

```
LD A,7
```

In other words, if you wish to learn machine code programming, get an assembler.

It is worth pointing-out at this stage that machine code instructions can occupy more

Computers are stupid – when programming you will appreci processors, such as the Z-80 rudimentary mathematics su techniques are required to ext

than one consecutive byte in memory. The LD register direct instruction takes two locations; the first byte is used to identify the register and the second byte is for identifying the instruction.

The second method of loading a register is to load it with the contents of another register, e.g., LET A=B in Basic. The third method of assigning a register is to assign it a value from a memory location. That is similar to the Basic LET A= PEEK(625).

ge

The difficulty with single-byte registers is that they can have values only between 0 and 255. That is sufficient for most purposes but if a bigger number needs to be stored, individual registers can be treated as pairs. In other words, registers B and C are register pair BC. The range then covers 0 to 65535.

It is important to note that you cannot use registered pairs and single registers in the same program, e.g.:

```
LD B,6
LD C,10
LD BC,3456
```

and hope that BC is different from B and C – they are the same registers but paired.

The 16-bit general registers cannot be loaded with the contents of another register or with the contents of as memory location directly. It is possible, however, to use the registered pair HL as a pointer in memory:

```
LD HL,16377
LD C,(HL)
```

In that example, the register pair HL is loaded with 16377

you start to learn machine code te just how silly they are. Eight-bit nd 6502, can only perform n as add and subtract. Special nd their repertoire of operations.

and the contents of memory location 16377 are then loaded into register C. The LD C,(HL) is equivalent to LET C=PEEK (HL) in Basic.

There are a few ways in which program control can be changed in a machine code program. The Z-80 has two types of GOTO which are called JP. The first is very much like a GOTO but instead of jumping to a line number a memory address is specified:

```
JP 3432
```

That transfers control to memory location 3432. It can also be used in conjunction with the HL register pair:

```
LD HL,5452
JP (HL)
```

The JR, or Jump Relative instruction, will move the program pointer (PC) backward 128 or forward 127 bytes. The instruction is two bytes long, so to jump back 10 bytes the following code could be used:

```
16000 LD A,7
      .
      .
      .
16008 JR –10
```

Another method of changing the program flow is to make use of the program counter. The PC always holds the address of the next instruction to be executed. Other Z-80 instructions can alter the contents of the PC and thus alter the program flow. Two such instructions are CALL and RET which are equivalent to the Basic GOSUB and RETURN.

Computers are stupid and when you start to learn machine code you will appreciate how silly they are. An 8-bit processor such as the Z-80 or 6502 CPU can perform only rudimentary mathematics such as add and subtract.

The accumulator is very important, as it is the only 8-bit register which can be used to store the result of a mathematics operation. The contents of the common registers, though, can be incremented or decremented by one:

Even when you are not running a program, your computer is far from idle. It is executing thousands of machine code instructions per second.

```
INC L
```

That adds one to the contents of register L. DEC operates in a similar manner, insofar as it subtracts one from a register. Adding and subtracting from the accumulator is also fairly straightforward:

```
ADD A,54
```

That adds 54 to the contents of the accumulator. To subtract a number from the accumulator, SUB n is used, as it is pointless specifying the accumulator in the instruction as it is the only register from which there can be subtraction.

There are a few more instructions in the Z-80 instruction set and most of them have different modes of operation. As can be gathered from this brief introduction to assembly language, all machine code instructions perform very simple tasks. To perform an operation such as finding the cube of a number would take several lines of assembly code.

If it takes several lines to do such a small task, you could be forgiven for thinking that writing a game may take a long time. That is not the case. All micros have an operating system and Basic which are made up of dozens of self-containing machine code routines which perform functions such as printing to the screen, detecting keypresses and so on. Using those routines it is possible to write very powerful programs which do not occupy too much room.

The best way to learn machine code is by practice. I would not recommend POKing decimal numbers in your machine. Instead, buy a good assembler and dissassembler – a program which converts the binary numbers into assembly language.

Another worthwhile purchase – as well as a good book on the subject – is a machine code monitor. That type of program allows you to examine and alter the contents of memory at leisure.

Musical pictures

Probably the most exciting programming task you can undertake with your micro is getting it to generate impressive music and pictures. With the comparatively large memory capacities of many of home micros, animated cartoon-style sequences with accompanying soundtracks are possible.

In the next six issues of this programming course, we will take you through the fundamentals of music and graphics programming and, near the end of the course, show you how to bring music and graphics together in one program. The example programs will be modular, so that combining music and graphics in the later stages of the course will be made easier.

There are a number of popular methods of developing graphics for use in micro software. The most difficult but by far the most common is to map out you computer screen pixel by pixel, determining what colour each would be. They are determined most commonly when programming in Basic by the use of data statements, which fill in the numbers which correspond to the information about colour and location of pixels.

The other less common but much easier method of incorporating graphics into your program would be considered as cheating by many programming purists but it is worth considering. It involves using a dedicated graphics development package which allows you to employ a joystick, mouse or cursor keys to draw pictures on-screen.

Once you have drawn those pictures using such software, you save them to tape, disc or Microdrive as picture files which can then be called-up and incorporated into your Basic programs. Pictures developed in such a manner are best used in a static fashion, such as providing graphic representations of events in a text adventure game.

We will be looking at both methods of incorporating graphics into your software, starting with the former more conventional method. Before you can begin developing colour graphics on your micro, you should get a good idea of what resources are available to you – i.e., the colours. Figure one is a short program for the Sinclair Spectrum which shows the basic eight BORDER colours on the machine.

Figure two is a similar program for the Commodore 64 which generates a series of colours randomly when you hit the space bar/key, giving you a quick idea of the colours available.

Sound, speech and music features on home micros vary enormously. The music facilities on the Spectrum Plus have been criticised as 'weedy', although many excellent add-ons are available to bring it to full strength, while the Commodore and BBC micro music generation systems have long been highly-praised.

In the next six months, we will look at ways of bringing out the most of each of those features on your machine, both through judicious programming and by buying certain articles of hardware which will enhance your system. We will give you specifications on Midi music interfaces which allow you to fit piano-type keyboards to your micro, synthesiser programs designed to let you simulate a wide variety of musical instruments, and a series of program listings which will give you a DIY method of developing your own music system.

To start, we offer a small sample listing – figure three – for the Commodore 64 which lets you play the notes C, D, E, F, G, A, B and 'high' C – using the keys Q, W, E, R, T, Y, U and I respectively.

In the next issue, we will look at music on the Amstrad, music interfaces for the Commodore 64, the BBC sound chip and speech systems on the Spectrum.

```
5 REM COLOURS EXAMPLE FOR THE SINCLAIR SPECTRUM
10 CLS
20 FOR X=0 TO 7
30 BORDER X
40 FOR Q=1 TO 200:NEXT Q
50 NEXT X
60 GOTO 10
```

Figure 1.

```
5 REM COMMODORE 64 COLOURS EXAMPLE
10 GET B$:IF B$="" THEN GOTO 10
20 PRINT CHR$(147)
30 X=INT(RND(0)*15)
40 IF X=0 THEN GOTO 30
50 POKE 53281,X
60 POKE 53260,X
70 FOR Q=1 TO 100:NEXT Q
80 GOTO 10
```

Figure 2.

```
10 PRINT CHR$(147)
20 GOSUB 450
30 SK=1
40 PRINT CHR$(147)
50 PRINT "PRESS THE 'A' KEY TO PLAY A TUNE"
60 C=-1
70 GET A$:IF A$="" THEN GOSUB 1000
80 GOSUB 600
90 GOTO 40
100 REM READ THE KEYBOARD
110 GET K$:IF K$="" THEN 110
120 F1=(K$="Q")*N(0)+(K$="2")*N(1)+(K$="W")*N(2)+
    (K$="3")*N(3)+(K$="E")*N(4)
130 F1=(K$="R")*N(5)+(K$="5")*N(6)+(K$="T")*N(7)+
    (K$="6")*N(8)+(K$="Y")*N(9)+F1
140 F1=(K$="7")*N(10)+(K$="U")*N(11)+(K$="I")*N(12)+F1
150 IF F1=0 THEN 110
160 F1=-F1
170 GOSUB 600
180 POKE I,F1-INT(F1/256)*256
190 POKE H,INT(F1/256)
200 RETURN
450 REM CLEAR SOUND CHIP
460 V=54296:AD=54277:SR=54278:W=54276
470 GOSUB 600
480 H=54273:I=54272
490 DIM N(12),S(30)
500 FOR T=0 TO 12:READ A:N(T)=A:NEXT
510 DATA 2195, 2325, 2463, 2630, 2795, 2930, 3104, 3288,
    3484, 3691, 3910, 4142, 4389
520 RETURN
600 FOR B=0 TO 9:POKE B+54272,0:NEXT B
610 POKE V,15:POKE AD,10:POKE SR,0:POKE W,17
620 RETURN
1000 GOSUB 110
1010 GOTO 1000
```

Figure 3.

NEXT MONTH

Delving deeper into machine code music on the Amstrad .

E for EPROM to H for Hardware and everything between in our Glossary.

At home with Pascal.